**Platform Dependent Compilation**

Unity includes a feature named "Platform Dependent Compilation". This consists of some preprocessor directives that let you *partition* your scripts to compile and execute a section of code exclusively for one of the supported platforms.

Furthermore, you can run this code within the Editor, so you can compile the code specifically for your mobile/console and test it in the Editor!

**Platform Defines**

The platform defines that Unity supports for your scripts are:

|  |  |
| --- | --- |
|  |  |
| **UNITY\_EDITOR** | Define for calling Unity Editor scripts from your game code. |
| **UNITY\_STANDALONE\_OSX** | Platform define for compiling/executing code specifically for Mac OS (This includes Universal, PPC and Intel architectures). |
| **UNITY\_DASHBOARD\_WIDGET** | Platform define when creating code for Mac OS dashboard widgets. |
| **UNITY\_STANDALONE\_WIN** | Use this when you want to compile/execute code for Windows stand alone applications. |
| **UNITY\_STANDALONE\_LINUX** | Use this when you want to compile/execute code for Linux stand alone applications. |
| **UNITY\_STANDALONE** | Use this to compile/execute code for any standalone platform (Mac, Windows or Linux). |
| **UNITY\_WEBPLAYER** | Platform define for web player content (this includes Windows and Mac Web player executables). |
| **UNITY\_WII** | Platform define for compiling/executing code for the Wii console. |
| **UNITY\_IPHONE** | Platform define for compiling/executing code for the iPhone platform. |
| **UNITY\_ANDROID** | Platform define for the Android platform. |
| **UNITY\_PS3** | Platform define for running PlayStation 3 code. |
| **UNITY\_XBOX360** | Platform define for executing Xbox 360 code. |
| **UNITY\_NACL** | Platform define when compiling code for Google native client (this will be set additionally to **UNITY\_WEBPLAYER**). |
| **UNITY\_FLASH** | Platform define when compiling code for Adobe Flash. |
| **UNITY\_BLACKBERRY** | Platform define for a Blackberry10 device. |
| **UNITY\_WP8** | Platform define for Windows Phone 8. |
| **UNITY\_METRO** | Platform define for Windows Store Apps (additionally **NETFX\_CORE** is defined when compiling C# files against .NET Core). |

Also you can compile code selectively depending on the version of the engine you are working on. Currently the supported ones are:

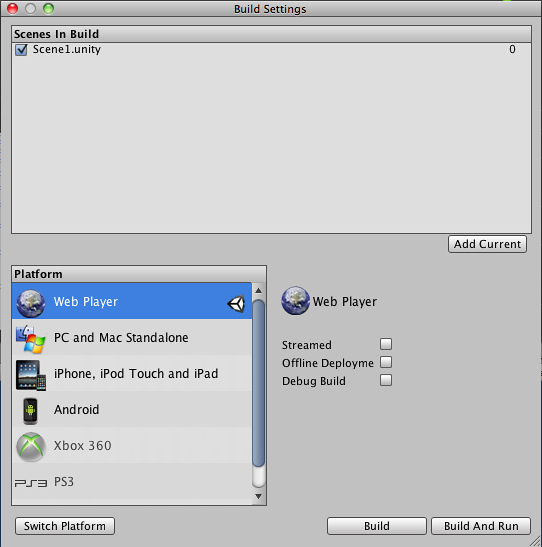
|  |  |
| --- | --- |
| **UNITY\_2\_6** | Platform define for the major version of Unity 2.6. |
| **UNITY\_2\_6\_1** | Platform define for specific version 1 from the major release 2.6. |
| **UNITY\_3\_0** | Platform define for the major version of Unity 3.0. |
| **UNITY\_3\_0\_0** | Platform define for the specific version 0 of Unity 3.0. |
| **UNITY\_3\_1** | Platform define for major version of Unity 3.1. |
| **UNITY\_3\_2** | Platform define for major version of Unity 3.2. |
| **UNITY\_3\_3** | Platform define for major version of Unity 3.3. |
| **UNITY\_3\_4** | Platform define for major version of Unity 3.4. |
| **UNITY\_3\_5** | Platform define for major version of Unity 3.5. |
| **UNITY\_4\_0** | Platform define for major version of Unity 4.0. |
| **UNITY\_4\_0\_1** | Platform define for major version of Unity 4.0.1. |
| **UNITY\_4\_1** | Platform define for major version of Unity 4.1. |
| **UNITY\_4\_2** | Platform define for major version of Unity 4.2. |

**Note:** For versions before 2.6.0 there are no platform defines as this feature was first introduced in that version.

**Testing precompiled code.**

We are going to show a small example of how to use the precompiled code. This will simply print a message that depends on the platform you have selected to build your target.

First of all, select the platform you want to test your code against by clicking on **File -> Build Settings**. This will bring the build settings window to select your target platform.

  
*Build Settings window with the WebPlayer Selected as Target platform.*

Select the platform you want to test your precompiled code against and press the **Switch Editor** button to tell Unity which platform you are targeting.

Create a script and copy/paste this code:-

// JS

function Awake() {

#if UNITY\_EDITOR

Debug.Log("Unity Editor");

#endif

#if UNITY\_IPHONE

Debug.Log("Iphone");

#endif

#if UNITY\_STANDALONE\_OSX

Debug.Log("Stand Alone OSX");

#endif

#if UNITY\_STANDALONE\_WIN

Debug.Log("Stand Alone Windows");

#endif

}

// C#

using UnityEngine;

using System.Collections;

public class PlatformDefines : MonoBehaviour {

void Start () {

#if UNITY\_EDITOR

Debug.Log("Unity Editor");

#endif

#if UNITY\_IPHONE

Debug.Log("Iphone");

#endif

#if UNITY\_STANDALONE\_OSX

Debug.Log("Stand Alone OSX");

#endif

#if UNITY\_STANDALONE\_WIN

Debug.Log("Stand Alone Windows");

#endif

}

}

// Boo

import UnityEngine

class PlatformDefines (MonoBehaviour):

def Start ():

ifdef UNITY\_EDITOR:

Debug.Log("Unity Editor")

ifdef UNITY\_IPHONE:

Debug.Log("IPhone")

ifdef UNITY\_STANDALONE\_OSX:

Debug.Log("Stand Alone OSX")

ifdef not UNITY\_IPHONE:

Debug.Log("not an iPhone")

Then, depending on which platform you selected, one of the messages will get printed on the Unity console when you press play.

Note that in c# you can use a CONDITIONAL attribute which is a more clean, less error-prone way of stripping out functions, see [http://msdn.microsoft.com/en-us/library/4xssyw96.aspx](http://msdn.microsoft.com/en-us/library/4xssyw96v=vs.90.aspx).

In addition to the basic *#if* compiler directive, you can also use a multiway test in C# and JavaScript:-

#if UNITY\_EDITOR

Debug.Log("Unity Editor");

#elif UNITY\_IPHONE

Debug.Log("Unity iPhone");

#else

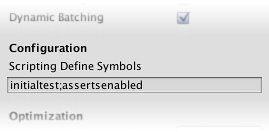
Debug.Log("Any other platform");

#endif

However, Boo currently supports only the *ifdef* directive.

**Platform Custom Defines**

It is also possible to add to the built-in selection of defines by supplying your own. In the **Other Settings** panel of the [Player Settings](http://docs.unity3d.com/Documentation/Components/class-PlayerSettings40.html), you will see the Scripting Define Symbols textbox.



Here, you can enter the names of the symbols you want to define for that particular platform, separated by semicolons. These symbols can then be used as the conditions for *#if* directives just like the built-in ones.

**Global Custom Defines**

You can define your own preprocessor directives to control which code gets included when compiling. To do this you must add a text file with the extra directives to the "Assets/" folder. The name of the file depends on the language you are using, and the extension is **.rsp**:

|  |  |
| --- | --- |
| C# | <Project Path>/Assets/smcs.rsp |
| C# - Editor Scripts | <Project Path>/Assets/gmcs.rsp |
| UnityScript | <Project Path>/Assets/us.rsp |
| Boo | <Project Path>/Assets/boo.rsp |

As an example, if you include the single line "-define:UNITY\_DEBUG" in your **smcs.rsp** file the define UNITY\_DEBUG will exist as a global define for C# scripts, except for Editor scripts.

Every time you make changes to .rsp files you will need to recompile for them to be effective. You can do this by updating or reimporting a single script (.js, .cs or .boo) file.

If you want to modify only global defines, you should use **Scripting Define Symbols** in [Player Settings](http://docs.unity3d.com/Documentation/Components/class-PlayerSettings40.html), because this will cover all the compilers. If you choose the .rsp files instead, you'll have to provide one file for every compiler Unity uses, and you won't know when one or another compiler is used.

The use of the .rsp files is described in the help section of the **smcs** application which is included in the Editor installation folder. You can get more information by running "smcs -help". Also, bear in mind the .rsp file needs to match the compiler being invoked. For example, when targeting the web player, **smcs** is used with smcs.rsp; when targeting standalone players, **gmcs** is used with gmcs.rsp; when targeting MS compiler, **csc** is used with csc.rsp; and so on.

<http://docs.unity3d.com/Documentation/Manual/PlatformDependentCompilation.html>